

Algorithms for Uncertainty Quantification

Tutorial 1: Python overview

In this worksheet, we do a short overview of programming in `python` and we show how to install the uncertainty quantification library `chaospy`. Note: Everything that follows is a lot easier if you use a Unix-based operating system on your machine.

We assume that you are familiar with the basics concepts of computer programming:

- data types
- variables
- loops
- arrays
- functions
- etc.

If you are new to programming and you want to cover the basic concepts, see e.g. tutorialspoint.com/computer_programming/computer_programming_basics.htm.

Short overview of programming with python

Throughout these tutorials, we use `python 2.7`. How to use `python`¹?

- directly in the command line: simply type `python`

¹extensive `python 2.7` documentation at <https://docs.python.org/2/index.html>

- via Jupyter² notebooks
- via an IDE, e.g. Spyder, Eclipse + PyDev, PyCharm

python is

- strongly typed: types are enforced
- dynamically, implicitly typed: you don't need to declare variables
- case sensitive: a and A are different variables
- object-oriented: everything is an object

Documentation

Via

`help(<object>)`, `<object>.__doc__`

E.g.

```
>>> ['a']. __doc__
>>> "list() -> new empty list\nlist(iterable) -> new list\n    initialized from iterable's items"
```

Data structures

Build-in data structures

- lists: similar to 1D arrays
- dictionaries: hash-tables (key-value pairs)
- tuples: immutable 1D arrays
- sets: since python 2.5

²<http://jupyter.org/>

```

>>> # list example
>>> mylist = ['a', 1, 2, [1, 2]]
>>> mylist[1] = 22
>>> mylist
[1, 22, 'a', [1, 2]]
>>> # tuple example
>>> mytuple = (1, 2)
>>> mytuple[1] = 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # dictionary example
>>> mydict = {1:'a', 0:'b'}
>>> mydict[0]
'b'
>>> mydict.keys()
[0, 1]
>>> mydict.values()
['b', 'a']
>>> mydict[5] = 'foo'
>>> mydict
{0: 'a', 1: 'b', 5: 'foo'}
>>> # set example
>>> mylist = [1, 2, 2, 3]
>>> set(mylist)
set([1, 2, 3])
>>> # list slicing
>>> mylist = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> mylist[0:-1]
[1, 2, 3, 4, 5, 6, 7, 8]
>>> mylist[2:4]
[3, 4]
>>> mylist[-1:-4:-1]
[9, 8, 7]
>>> mylist[:5]
[1, 2, 3, 4, 5]

```

Flow control statements

Flow control statements in python: `if`, `for`, and `while`. To obtain a list of numbers, use `range<number>` (or better, `xrange<number>`)

```
>>> for i in xrange(5):
...     if i % 2:
...         print i
...     else:
...         print i - 1
...
-1
1
1
3
3
```

Functions

Functions are declared with the `def` keyword. Optional arguments are set in the function declaration. In python, functions can return a tuple (this way, you can effectively return multiple values)

```
>>> def increase_by_one(x):
...     return x + 1
...
>>> increase_by_one(2)
3
>>> increase_by_one(2.3)
3.3
>>> new_increase_by_one = lambda x: (x + 1, 0)
>>> res = new_increase_by_one(1)
>>> res
(2, 0)
>>> res[1] = 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Use of external libraries

```
>>> # import entire library
>>> import [libname]
>>> # import specific functions/packages
>>> from [libname] import [funcname|packagename]
```

Object-oriented programming

Classes are declared with the `class` keyword. Private variables and methods can be declared, however, by convention, this is not enforced by the language.

```
>>> class MyClass:
...     variable = 10
...     def __init__(self):
...         print 'this is an instance of MyClass'
...     def myfunction(self, x):
...         return 2*x
...
>>> instance = MyClass()
this is an instance of MyClass
>>> instance.myfunction(10)
20
>>> instance.variable
10
>>> # inheritance example
>>> class SubClass(MyClass):
...     def __init__(self):
...         print 'this is an instance of the subclass'
...
>>> instance=SubClass()
this is an instance of the subclass
```

Polymorphism is also possible

```
>>> class Animal:
...     def __init__(self, name):
...         self.name = name
...     def talk(self):
...         raise NotImplementedError("Abstract method must
...         be implemented")
```

```

...
>>> class Cat(Animal):
...     def talk(self):
...         return 'Meow!'
...
>>> class Dog(Animal):
...     def talk(self):
...         return 'Woof!_Woof!'
...
>>> class Human(Animal):
...     def bark(self):
...         print 'UQ_is_the_best'
...
>>> animals = [Cat('Mr._Purrify'), Dog('Lassie'), Human('me')]
>>> for animal in animals:
...     print animal.name + ':_' + animal.talk()
...
Mr. Purrify: Meow!
Lassie: Woof! Woof!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
  File "<stdin>", line 5, in talk
NotImplementedError: Abstract method must be implemented

```

Scientific computing with python

Since `python` is an interpreted and dynamically typed programming language, `python` programs can be slow compared to compiled statically typed programming languages, such as C and Fortran.

However, using suitable libraries, `python` can be used for scientific computing³.

Numpy - multidimensional data arrays

package that provides high-performance vector, matrix and higher-dimensional data structures for `python`.

```
>>> import numpy as np
```

³some materials on scientific computing <https://github.com/jrjohansson/scientific-python-lectures>

```
>>> # do clever things
```

SciPy - Library of scientific algorithms

It includes

- Special functions (scipy.special)
- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Sparse Eigenvalue Problems (scipy.sparse)
- Statistics (scipy.stats)
- Multi-dimensional image processing (scipy.ndimage)
- File IO (scipy.io)

matplotlib - 2D and 3D plotting

Using Fortran and C code

In python it is relatively easy to call out to libraries with compiled C or Fortran code

- Fortran: F2PY
- C: ctypes, Cython

Tools for high-performance computing applications

- thread parallelism: the multiprocessing package
- IPython parallel
- MPI: mpi4py
- opencl: pyopencl
- etc.

Assignment

A linear damped oscillator is modeled by a second order ODE

$$\begin{cases} \frac{d^2y}{dt^2}(t) + c\frac{dy}{dt}(t) + ky(t) = f \cos(\omega t) \\ y(0) = y_0 \\ \frac{dy}{dt}(0) = y_1, \end{cases} \quad (1)$$

where c is the damping coefficient, k the spring constant, f the forcing amplitude, ω the frequency, y_0 represents the initial position, whereas y_1 is the initial velocity.

Assignment 1

Transform the above equation in a first order system of ODEs. Hint: Use helper functions

$$\begin{aligned} z_0(t) &:= y(t), \\ z_1(t) &:= \frac{dy(t)}{dt}, \\ &\vdots \\ z_{n-1}(t) &:= \frac{dy^{n-1}(t)}{dt^{n-1}} \end{aligned}$$

and transform into system

$$\begin{aligned}\frac{dz_0}{dt} &:= z_1, \\ \frac{dz_1}{dt} &:= z_2, \\ &\vdots \\ f(t, z_0, \frac{dz_0}{dt}, \frac{dz_1}{dt}, \dots, \frac{dz_{n-1}}{dt}) &:= 0.\end{aligned}$$

Assignment 2

Considering $t \in [0, 20]$, $c = 0.5$, $k = 2.0$, $f = 0.5$, $\omega = 1.05$, $y_0 = 0.5$, $y_1 = 0.0$ and the setup as in Assignment 1, discretize the oscillator using explicit Euler time on the system with $\Delta t = 0.1, 0.05, 0.01$. You can use standard python lists or numpy lists. Change $\Delta t = 0.5$, what do you observe?

Assignment 3

With the same setup, solve (1) using the `odeint`⁴ function from `scipy.integrate`.

chaospy installation/building from source

Throughout these tutorials, we will use the `chaospy` library. `chaospy`⁵ is a library designed to perform uncertainty quantification using (but not restricted to) polynomial chaos expansions and advanced Monte Carlo methods implemented in Python 2 and 3. Assuming that you have a Unix-based operating system on your machine, to install `chaospy` you could use the `pip` package management system

- install `pip`: `sudo apt install python-pip`
- install `numpy`, `scipy`, `matplotlib`: `pip install numpy scipy matplotlib`⁶
- install `scikit-learn`⁷: `pip install scikit-learn`

⁴see <https://docs.scipy.org/doc/scipy-0.17.0/reference/generated/scipy.integrate.odeint.html>

⁵online documentation <http://chaospy.readthedocs.io/en/master/>

⁶on systems where you don't have root access, use `pip install --user`

⁷to support more regression methods in `chaospy`; optional

- install chaospy: `(sudo)8 pip install chaospy`

or you could build chaospy directly from source

- install git: `sudo apt install git`
- clone the chaospy repository: `git clone https://github.com/jonathf/chaospy.git`
- build chaospy
 - `cd /path_to_chaospy/chaospy`
 - `sudo apt install python-numpy python-scipy python-matplotlib9`
 - `sudo apt install python-scikit-learn`
 - `(sudo)10 python setup.py install`
 - add `/path_to_chaospy/chaospy/build` to your `PYTHONPATH`
 - * type `export $PYTHONPATH=/path_to_chaospy/chaospy/build:$PYTHONPATH` in your terminal or
 - * add `$PYTHONPATH=/path_to_chaospy/chaospy/build:$PYTHONPATH` into your `.bashrc` file
- at the end test the build locally
 - `pip install -r requirements-dev.txt`
 - `python setup.py test`

chaospy installation on Windows

Disclaimer: this was not tried out, hence, some steps might be needed to be adjusted

- download and install anaconda: following the steps from <https://docs.continuum.io/anaconda/install-windows.html>
- install numpy, scipy, matplotlib via conda `install numpy scipy matplotlib`
- install scikit-learn, optional via conda `install scikit-learn`

⁸it might work without root access

⁹alternatively, you could use `pip install -r requirements-dev.txt`

¹⁰it might work without root access

- add `\path\to\anaconda\lib` to the `pythonpath` or
- pipe every `.py` file to anaconda's python (in cmd):
`assoc .py=Python.File`
`ftype Python.File=C:path\to\Anaconda\python.exe "%1" %*`